AMENDMENTS TO THE CLAIMS

1. (Currently Amended) A method, comprising:

encountering a function call instruction that calls a called function during program execution;

saving a return address in a first stack memory and in a second stack memory at the same time, the return address containing an instruction to be executed after execution of the called function, wherein the second stack memory stores a return address for each function call instruction that calls a called function during program execution;

executing the called function;

determining if the return address stored in the first stack memory matches the return address stored in the second stack memory to provide protection from a buffer overflow attack;

executing exception handling code if an exception is generated when the return addresses do not match,

wherein the exception handling code determines what value to pass to a program pointer based on the return address retrieved from each of the first and second ~~stacks~~ stack memories.

2. – 4. (Cancelled)

5. (Previously Presented) The method of claim 1, wherein the exception handling code terminates execution of the program.

6. (Currently Amended) A method, comprising:

processing instructions within a virtual machine;

saving a return address in a first stack memory and in a second stack memory at the same time, the return address being an address at which program execution is to resume after execution of a called function;

comparing the return addresses saved in the first and second stack memories upon execution of the called function; and

exiting the virtual machine if the return addresses do not match to provide protection from a buffer overflow attack wherein the second stack memory stores a return address for each

function call instruction that calls a called function during program execution, and wherein an exception handler determines if the return address from the first stack memory or the return address from the second stack memory is to be used as a value for an instruction pointer.

7. (Previously Presented) The method of claim 6, further comprising passing control to the exception handler.

8. (Cancelled)

9. (Currently Amended) A method, comprising:

creating first and second ~~stacks~~ stack memories for a program during execution of the program;

encountering a function call to a called function;

storing data for the called function and a return address in the first stack memory;

storing the return address in the second stack memory at the same time as the first stack memory; and

passing control of the program to an exception handler if the return address stored in the first stack memory does not match the return address stored in the second stack memory upon execution of the called function to provide protection from a buffer overflow attack, wherein the exception handler determines if the return address from the first stack memory or the return address from the second stack memory is to be used as a value for an instruction pointer~~, wherein the exception handler determines if the return address from the first stack, or the return address from the second stack is to be used as a value for an instruction pointer~~.

10. (Cancelled)

11. (Currently Amended) A processor, comprising:

memory management logic to allocate first and second memory locations corresponding to first and second ~~stacks~~stack memories, respectively, when a function call instruction calls to a called function is encountered during program execution;

function call logic to write a return address to a memory location from the first memory locations and to a memory location from the second memory locations at the same time, the return address being an address at which program flow is to resume after execution of the called function; and

buffer overflow control logic to determine if the return address retrieved from the first memory locations matches the return address retrieved from the second memory locations, upon execution of the called function to provide protection from a buffer overflow attack wherein the exception handler determines if the return address from the first stack memory, or the return address from the second stack memory is to be used as a value for an instruction pointer.

12. (Previously presented) The processor of claim 11, wherein the function call logic and the buffer overflow control logic comprises microcode stored within the processor.

13. (Currently Amended) A system, comprising:

a memory; and

a processor coupled to the memory, the processor comprising memory management logic to allocate first and second memory locations corresponding to first and second ~~stacks~~stack memories, respectively, when a function call instruction that calls a called function is encountered during program execution;

function call logic to write a return address to a memory location from the first memory locations and to a memory location from the second memory locations at the same time, the return address being an address at which program flow is to resume after execution of the called function; and

buffer overflow control logic to determine if the return address retrieved from the first memory locations matches the return address retrieved from the second memory locations, upon execution of the called function to provide protection from a buffer overflow attack wherein the exception handler determines if the return address from the first stack memory, or the return address from the second stack memory is to be used as a value for an instruction pointer.

14. (Previously presented) The system of claim 13, wherein the memory management logic, the function call logic, and the buffer overflow control logic comprise microcode stored within the processor.

15. (Currently Amended) A computer ~~readable~~ recordable type medium having stored thereon a sequence of instructions which when executed by a processor, cause the processor to perform a method comprising:

encountering a function call instruction that calls a called function during program execution;

saving a return address in a first stack and in a second stack at the same time, the return address containing an instruction to be executed after execution of the called function;

executing the called function; and

determining if the return address stored in the first stack matches the return address stored in the second stack to provide protection from a buffer overflow attack wherein the exception handler determines if the return address from the first stack, or the return address from the second stack is to be used as a value for an instruction pointer.

16. (Currently Amended) The computer ~~readable~~ recordable type medium of claim 15, wherein the method further comprises generating an exception if the return addresses do not match.

17. (Currently Amended) A computer recordable type medium having stored thereon a sequence of instructions which when executed by a processor, cause the processor to perform a method comprising:

processing instructions within a virtual machine;

saving a return address in a first stack memory and in a second stack memory at the same time, the return address being an address at which program execution is to resume after execution of a called function;

comparing the return addresses saved in the first and second stack memories upon execution of the called function; and

exiting the virtual machine if the return addresses do not match to provide protection from a buffer overflow attack wherein the exception handler determines if the return address from the first stack memory, or the return address from the second stack memory is to be used as a value for an instruction pointer.

18. (Previously Presented) The computer recordable type medium of claim 17, wherein the method further comprises passing control to an exception handler.

19. (Currently Amended) A computer recordable type medium having stored thereon a sequence of instructions which when executed by a processor, cause the processor to perform a method comprising:

creating first and second ~~stacks~~ stack memories for a program during execution of the program;

encountering a function call to a called function;

storing data for the called function and a return address in the first stack memory;

storing the return address in the second stack memory at the same time as the first stack memory; and

passing control of the program to an exception handler if the return address stored in the first stack memory does not match the return address stored in the second stack memory upon execution of the called function to provide protection from a buffer overflow attack wherein the exception handler determines if the return address from the first ~~stack,~~stack memory or the return address from the second stack memory is to be used as a value for an instruction pointer~~, wherein the exception handler determines if the return address from the first stack and the return address from the second stack is to be used as a value for an instruction pointer~~.

20. (Cancelled)